CS 61B Spring 2025

Asymptotics II

Discussion 05: February 24, 2025

1 I am Speed

(a) For each code block below, fill in the blank(s) so that the function has the desired runtime. Do not use any commas. If the answer is impossible, just write "impossible" in the blank. Assume that `System.out.println` runs in constant time. You may use Java's `Math.pow(x, y)` to raise `x` to the power of `y`.

```
// Desired Runtime: Theta(N)
public static void f1(int N) {
    for (int i = 1; i < N; ____________){
        System.out.println("hi Dom");
    }
}
```

Solution:

```
// Desired Runtime: Theta(N)
public static void f1(int N) {
    for (int i = 1; i < N; i += 1){
        System.out.println("hi Dom");
    }
}
```

Note the solution could be `i += C`, where `C` is some constant independent of `N`. This is because even if we did for example, `i += 10`, we would do $\frac{N}{10}$ work in total, which is still $\Theta(N)$.

```
// Desired Runtime: Theta(log N)
public static void f2(int N) {
    for (int i = 1; i < N; ____________) {
        System.out.println("howdy Ergun");
    }
}
```

Solution:

```
// Desired Runtime: Theta(log N)
public static void f2(int N) {
    for (int i = 1; i < N; i *= 2) {
        System.out.println("howdy Ergun");
    }
}
```

Here, the solution could be `i *= C`, where `C` is some constant independent of `N`. This is because even if we did for example, `i *= 5`, we would do $\log_5 N$ work in total, and in general $\log_i N$ work, which is still $O(\log n)$.

```
// Desired Runtime: Theta(1)
public static void f3(int N) {
```

```
    for (int i = 1; _____________; i += 1) {
        System.out.println("hello Anniyat");
    }
}
```

Solution:

```
// Desired Runtime: Theta(1)
public static void f3(int N) {
    for (int i = 1; i < 1000}; i += 1) {
        System.out.println("hello Anniyat");
    }
}
```

Again, the solution is actually just `i < C`, where `C` is some constant independent of the input `N`.

```
// Desired Runtime: Theta(2^N)
// This one is tricky! Hint: think about the dominating term in 1 + 2 + 4 + 8 + ...
+ f(N)
public static void f4(int N) {
    for (int i = 1; ___________________; i *= 2) {
        for (int j = 0; j < i; j += 1) {
            System.out.println("what's up Alyssa");
        }
    }
}
```

Solution:

```
// Desired Runtime: Theta(2^N)
// This one is tricky! Hint: think about the dominating term in 1 + 2 + 4 + 8 + ... + f(N)
public static void f4(int N) {
    for (int i = 1; i < Math.pow(2, N)}; i *= 2) {
        for (int j = 0; j < i; j += 1) {
            System.out.println("what's up Alyssa");
        }
    }
}
```

(b) *Extra:* Give the worst case and best case running time in $\Theta(\cdot)$ notation in terms of M and N. Assume that `kachow()` runs in $\Theta(N^2)$ time and returns a `boolean`.

```
for (int i = 0; i < N; i += 1) {
    for (int j = 1; j <= M; ) {
        if (kachow()) {
            j += 1;
        } else {
            j *= 2;
        }
    }
}
```

Solution: Worst case: $\Theta(N^3M)$, Best case: $\Theta(N^3 \log M)$
To see this, note that in the best case `kachow()` always returns `false`. Regardless of best/worst case, `kachow()` runs with each iteration of the inner loop. Hence `j` multiplies by 2 each inner loop iteration, which means the inner loop would take $N^2 \log M$ time for each iteration of the outer loop. In the worst case, `kachow()` always returns true, thus the inner loop iterates `M` times, each time calling `kachow()`, so the inner loop would take N^2M. Since the outer loop always iterates N times, we get the worst and best case by multiplying the time the inner loop takes by N.

2 Re-cursed with Asymptotics!

To help visualize the solutions better, a video walkthrough of this problem is linked here!

(a) What is the runtime of the code below in terms of `n`?

```
public static int curse(int n) {
    if (n <= 0) {
        return 0;
    } else {
        return n + curse(n - 1);
    }
}
```

$\Theta(n)$. On each recursive call, we do a constant amount of work. We make n recursive calls, because we go from n to 1. Then n recursive layers with 1 work at each layer is overall $\Theta(n)$ work.

(b) Can you find a runtime bound for the code below? We can assume the `System.arraycopy` method takes $\Theta(N)$ time, where N is the number of elements copied. The official signature is `System.arrayCopy(Object sourceArr, int srcPos, Object dest, int destPos, int length)`. Here, `srcPos` and `destPos` are the starting points in the source and destination arrays to start copying and pasting in, respectively, and `length` is the number of elements copied.

```
public static void silly(int[] arr) {
    if (arr.length <= 1) {
        System.out.println("You won!");
        return;
    }

    int newLen = arr.length / 2;
    int[] firstHalf = new int[newLen];
    int[] secondHalf = new int[newLen];

    System.arraycopy(arr, 0, firstHalf, 0, newLen);
    System.arraycopy(arr, newLen, secondHalf, 0, newLen);

    silly(firstHalf);
    silly(secondHalf);
}
```

Solution: $\Theta(N \log(N))$
At each level, we do N work, because the call to `System.arraycopy`. You can see that at the top level, this is N work. At the next level, we make two calls that each operate on arrays of length N/2, but that total work sums up to N. On the level after that, in four separate recursive function frames we'll call `System.arraycopy` on arrays of length N/4, which again sums up to N for that whole layer of recursive calls.

Now we look for the height of our recursive tree. Each time, we halve the length of N, which means that the length of the array N on recursive level k is roughly $N * \left(\frac{1}{2}\right)^k$. Then we will finally reach our base case $N \leq 1$ when we have $N * \left(\frac{1}{2}\right)^k = 1$. Doing some math, we see this can be transformed into $N = 2^k$, which means $k = \log_2(N)$. In other words, the number of layers in our recursive tree is $\log_2(N)$. If we have $\log_2(N)$ layers with $\Theta(N)$ work on each layer, we must have $\Theta(N \log(N))$ runtime.

(c) Given that `exponentialWork` runs in $\Theta(3^N)$ time with respect to input N, what is the runtime of `ronnie`?

```
public void ronnie(int N) {
    if (N <= 1) {
        return;
    }
    ronnie(N - 2);
    ronnie(N - 2);
    ronnie(N - 2);
    exponentialWork(N); // Runs in $Theta(3^N)$ time
}
```

$\Theta(3^N)$. Drawing out the recursive tree, the first level has $\Theta(3^N)$ work, the next level has $\Theta(3^{N-1})$ work, and so on until the last level which has approximately $\Theta\left(3^{\frac{N}{2}}\right)$ work. This gives the sum $\Theta(3^N) + \Theta(3^{N-1}) + ... + \Theta\left(3^{\frac{N}{2}}\right) = \Theta(3^N)$.

3 Asymptotics Proofs

As a reminder, the formal definitions of Ω, Θ, and O are provided below:

Let f, g be real-valued functions. Then:

$f(x) \in \Theta(g(x))$ if there exists $a, b, N_0 > 0$ such that for all $N > N_0$, $|ag(N)| \leq |f(N)| \leq |bg(N)|$.

$f(x) \in O(g(x))$ if there exists $b, N_0 > 0$ such that for all $N > N_0$, $|f(N)| \leq |bg(N)|$.

$f(x) \in \Omega(g(x))$ if there exists $a, N_0 > 0$ such that for all $N > N_0$, $|ag(N)| \leq |f(N)|$.

Informally, we say that $f(x) \in O(g(x))$ approximately means that $f(x) \leq g(x)$, and similarly, $f(x) \in \Theta(g(x))$ means $f(x) = g(x)$ and $f(x) \in \Omega(g(x))$ means $f(x) \geq g(x)$. This problem will explore why we can make this informal statement, by showing that the O relation shares many properties with the $\leq$ relation.

For this problem, let f, g, and h be real-valued functions, and let x, y, and z be real numbers. You won't be expected to write full proofs on exams, but this thinking style will be helpful on exams and especially in later classes.

(a) If $x \leq y$, then $y \geq x$. Show that if $f(x) \in O(g(x))$, then $g(x) \in \Omega(f(x))$

Solution: Assume x is large enough that the inequalities from all O bounds hold. From the definitions above we know that there exists $b > 0$ such that $|f(x)| \leq |bg(x)|$. Dividing both sides of the inequality by b, we get $|\frac{1}{b}f(x)| \leq |g(x)|$. If we define a to be $\frac{1}{b}$, this is the definition of $g(x) \in \Omega(f(x))$. Therefore if $f(x) \in O(g(x))$, $g(x) \in \Omega(f(x))$.

(b) If $x \leq y$ and $y \leq x$, then $x = y$. Show that if $f(x) \in O(g(x))$ and $g(x) \in O(f(x))$, then $f(x) \in \Theta(g(x))$

Solution: Assume x is large enough that the inequalities from all O bounds hold. Since $f(x) \in O(g(x))$, there must exist a value $b > 0$ such that $|f(x)| \leq |bg(x)|$. Additionally, since $g(x) \in O(f(x))$, there must exist a value $b' > 0$ such that $|g(x)| \leq |b'f(x)|$. Dividing both sides of the inequality by b', we get $|\frac{1}{b'}g(x)| \leq |f(x)|$. We then join the two inequalities on the shared $|f(x)|$ term to get $|\frac{1}{b'}g(x)| \leq |f(x)| \leq |bg(x)|$. If we define a to be $\frac{1}{b'}$, this is the definition of $f(x) \in \Theta(g(x))$.

(c) For any real number, $x \leq x$. Show that for any function, $f(x) \in O(f(x))$.

Solution: Assume x is large enough that the inequalities from all O bounds hold. From the definition of O, $f(x) \in O(g(x))$ if there exists $b, N_0 > 0$ such that $|f(x)| \leq |bg(x)|$ for all $x > N_0$. Let $g(x) = f(x)$. We see that for $|b| \geq 1$, $|f(x)| \leq |bf(x)|$ holds true regardless of the value of x. Since we were able to find $b, N_0 > 0$ such that the inequality was true, $f(x) \in O(f(x))$.

(d) If $x \leq y$ and $y \leq z$, then $x \leq z$. Show that if $f(x) \in O(g(x))$ and $g(x) \in O(h(x))$, then $f(x) \in O(h(x))$

Solution: Assume x is large enough that the inequalities from all O bounds hold. If $f(x) \in O(g(x))$, there exists $b > 0$ such that $|f(x) \leq |bg(x)|$. If $g(x) \in O(h(x))$, there exists b' such that $|g(x) \leq |b'h(x)|$. Substituting the second inequality into the first, we get that $|f(x) \leq |bb'h(x)|$. This is the definition of $f(x) \in O(h(x))$.

(e) For any pair of real numbers x and y, either $x < y$, $x = y$, or $x > y$. Show that this is NOT a property of O; that is, find functions f and g such that $f(x) \notin O(g(x))$ and $g(x) \notin O(f(x))$.

Solution: Any pair of functions that cannot consistently upper-bound each other would work here. An example of such a pair is $x * (1 + \sin(x))$ and $x * (1 + \cos(x))$.